Authentication & Authorization

Who are you and what can you do

Authentication

- Authentication refers to establishing an actor's identity sufficiently
  - Driver's license
  - University CatCard
  - Username and Password
- Only establishes identity
- Can be satisfied within the service, or through an external Identity Provider
**Authentication & Authorization**

**Authorization**

- Authorization refers to establishing what actions a verified actor can perform
  - Depends on Authentication
  - Many strategies
    - Groups
    - Roles
  - Usually dependent on the service / application to determine

**Authentication**

**Methods and Use Cases**

- There are usually different strategies for Authentication depending on if you are Authenticating a person, or some sort of other actor, like application code.
  - When you log in to D2L we use a different strategy (NetID+Password+DUO) than if you were authenticating to make certain API calls (Access Keys or Certificates)

**Authentication**

**Person Authentication**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Memorized Password</th>
<th>Password Manager</th>
<th>Password + MFA</th>
<th>Security Keys</th>
<th>Biometrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to use</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Always with you</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Never steal</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Security Level</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Recoverable</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Phishing resistant</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Doesn’t require shared secrets</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Authentication
HTTP Requests

- Username
- Password
- Path
- Query String

http://user:pass@example.com:80/path?query=yes#fragment

Authentication
HTTP Basic Authentication

- The username:password portion of a URL is translated into Basic Authentication by user agents (browsers, curl, etc)

GET /index.html HTTP/1.1
Host: example.com
Authentication: Basic dXNlcjpwYXNz

Authentication
HTTP Basic Authentication

- Basic Auth must only ever be used with TLS encrypted connections: HTTPS

http://user:pass@example.com/index.html
https://user:pass@example.com/index.html
**Authentication**

**HTTP Basic Authentication**

- Not encrypted, just base64 encoded

```python
import base64
username = "mark"
password = "aReallyGr8PasswordNoOneWillGuess"
authString = f"{username}:{password}" [base64.b64encode(authString.encode("UTF-8"))]
authHeader = f'Authentication: Basic {b64AuthString}"

print(authHeader)
# Prints the following
# Authentication: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
```

**Authentication**

**HTTP Basic Authentication**

- Libraries and tools make this really easy though

```python
import requests
url = "https://example.com/index.html"
username = "mark"
password = "aReallyGr8PasswordNoOneWillGuess"
response = requests.get(url, auth=(username, password))
headers = "
''.join(f"{k}: {v}") for k, v in response.request.headers.items()"

print(headers)
# User-Agent: python-requests/2.28.1
# Accept-Encoding: gzip, deflate
# Accept: */*
# Connection: keep-alive
# Authorization: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
```

**Authentication**

**HTTP Basic Authentication**

- Libraries and tools make this really easy though

```bash
$ curl -v --user "mark:aReallyGr8PasswordNoOneWillGuess" https://example.com/index.html
*   Trying 93.184.216.34:443...
* Connected to example.com (93.184.216.34) port 443 (#0)
* Server auth using Basic with user 'mark'
> GET /index.html HTTP/2
> Host: example.com
> authorization: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
> user-agent: curl/7.79.1
> accept: */*
> connection: keep-alive
>```

```python
url = "https://example.com/index.html"
username = "mark"
password = "aReallyGr8PasswordNoOneWillGuess"
response = requests.get(url, auth=(username, password))
headers = "
''.join(f"{k}: {v}") for k, v in response.request.headers.items()"

print(headers)
# User-Agent: python-requests/2.28.1
# Accept-Encoding: gzip, deflate
# Accept: */*
# Connection: keep-alive
# Authorization: Basic bWFyazphUmVhbGx5R3I4UGFzc3dvcmROb09uZVdpbGxHdWVzcw==
```
Authentication

HTTP Basic Authentication

- Libraries and tools make this really easy though

Authentication

Storing Usernames and Passwords

- How do you securely store passwords?
- Naive way is to just store the plaintext username and password in a data store. When someone logs in, you compare the password they entered with the one you stored.
  - Advantages:
    - You can see their passwords if they need to recover them
  - Disadvantages:
    - If you can see their passwords, so can the baddies (there are so many baddies)

Authentication

Storing Usernames and Passwords

- Better way is to use a strong hash algorithm with a salt
- Hashes are one-way transformation. Easy to transform an input into an output, but very very difficult to go the other way around.
- Store the hashed value in your data store
- Re-hash each password attempt, and compare the hashes
- If a baddie steals your data store hashes, your passwords are still relatively protected
- A salt value helps protect against pre-computed hash tables
import hashlib

username = "mark"
password = b"aReallyGr8PasswordNoOneWillGuess"

hashedPass = hashlib.sha512(password)
print(hashedPass.hexdigest())

hashedPass = hashlib.blake2b(password, salt=username)
print(hashedPass.hexdigest())

Authentication
Identity Providers
• Even better is to not get into the authentication business in the first place
• Use someone else’s set of identities
  • Social IdPs: Google, Facebook, Twitter, etc
  • Enterprise specific IdPs: University NetID
• Gets you off the hook for having to securely store authentication credentials

Authentication
Identity Providers
• Authentication Protocols
  • OAuth2
  • OpenID Connect (OIDC)
  • Security Assertion Markup Language (SAML)
  • Central Authentication Service (CAS)
Authentication
Central Authentication Service (CAS)

• CAS is pretty easy to implement ourselves
• Supported by the University's Shibboleth Identity Provider